Rendez-vous d'agents amnésiques†

Fabienne Carrier[1], Stéphane Devismes[1], Franck Petit[2] et Yvan Rivierre[1]

[1]*VERIMAG UMR 5104, Université Joseph Fourier, Grenoble (France),* `prenom.nom@imag.fr`
[2]*LIP6 UMR 7606, Université Pierre et Marie Curie (France),* `franck.petit@lip6.fr`

Dans cet article, nous présentons un algorithme déterministe de rendez-vous pour des agents évoluant dans un graphe non orienté anonyme quelconque. Les agents considérés sont autonomes, amnésiques et se déplacent de manière asynchrone. L'algorithme proposé est optimal en espace et asymptotiquement optimal en nombre de rondes.

Keywords: Agents amnésiques, Réseau d'agents mobiles, Optimalité, Rendez-vous

1 Introduction

Nous considérons *k agents* autonomes se déplaçant dans un *graphe non orienté* où les nœuds représentent des lieux et les arêtes les déplacements possibles entre ces lieux. Nous nous intéressons au problème du *rendez-vous*. Il existe plusieurs définitions de ce problème. Ici, nous considérons celle de [YY96] : un rendez-vous est un processus fini au terme duquel les *k* agents se retrouvent sur le même nœud. Initialement, le graphe ne comporte aucun agent. Les agents sont placés à des instants quelconques, dans n'importe quel ordre, sur n'importe quel nœud du graphe.

Nous étudions ce problème dans un modèle faible : les agents sont *amnésiques*, c'est-à-dire que leur mémoire volatile s'efface entre deux étapes de calcul. Leurs déplacements sont asynchrones. Ils ne peuvent ni se détecter les uns les autres, ni communiquer directement, même s'ils sont présents sur le même nœud. De plus, ils ne connaissent ni le graphe dans lequel ils évoluent, ni le nombre total d'agents. Enfin, les nœuds du graphe sont anonymes.

Il est montré dans [DMGK$^+$06] que sans hypothèse supplémentaire, ce problème n'admet pas de solution déterministe. Dans la suite, nous supposons donc que :

1. Chaque nœud dispose d'une mémoire locale finie appelée *tableau*. Un *tableau* peut être lu et écrit par tout agent présent sur le nœud.
2. Chaque nœud distingue ses arêtes incidentes au moyen d'*indices locaux*. Sans perte de généralité, chaque nœud v indice ses arêtes de 0 à $\delta_v - 1$, où δ_v est le degré de v. La fonction `Vient_de`$()$ permet à un agent de connaître l'indice local de l'arête par laquel il est arrivé sur un nœud. Quand un agent est placé initialement sur un nœud, il n'a traversé aucune arête, dans ce cas `Vient_de`$()$ retourne $\perp$.
3. Chaque agent possède une *étiquette*. Nous supposons une relation binaire $\prec$ telle qu'il existe une étiquette unique l, appelée *étiquette minimum*, qui satisfait : $l \prec l'$ pour toute autre étiquette l'. L'étiquette minimum est associée à *un seul agent*. En ce qui concerne les autres étiquettes, la relation $\prec$ n'est pas nécessairement cohérente : pour l' et l'' deux étiquettes distinctes non minimum, on a soit $l' \prec l''$, soit $l'' \prec l'$, ou les deux. Par ailleurs, toute étiquette non-minimum peut être associée à plusieurs agents. Un agent ne connaît pas *a priori* les étiquettes des autres. Un agent ne sait pas *a priori* si son étiquette est la plus petite ou non.

Notre contribution est la suivante : tout d'abord, dans la section 2, nous proposons un algorithme de rendez-vous. Nous présentons ensuite, dans la section 3, des bornes inférieures de complexité en espace et en temps (nombre de rondes) pour résoudre le rendez-vous dans notre modèle. Ces bornes montrent que notre algorithme de rendez-vous est optimal en espace et asymptotiquement optimal en nombre de rondes. A la fin de cette partie, nous justifions la nécessité de la plupart de nos hypothèses. Enfin, nous concluons et proposons quelques perspectives dans la section 4.

†Cet article est un résumé étendu de [CDPR09].

2 Algorithme

Nous proposons maintenant un algorithme de rendez-vous fonctionnant dans notre modèle. Chaque agent exécute l'algorithme *RDV* de la figure 1. Celui-ci consiste en un ensemble d'actions de la forme suivante : $\langle$ *nom* $\rangle$: $\langle$ *garde* $\rangle \rightarrow \langle$ *instruction* $\rangle$. Une *garde* est une expression booléenne portant sur les entrées de l'agent, à savoir son étiquette, la fonction `Vient_de`() et les variables du tableau du nœud sur lequel il se trouve. Une *instruction* est une séquence d'affectations des variables du tableau du nœud où l'agent se trouve, suivie éventuellement d'un déplacement (fonction `Aller_vers`()) vers un autre nœud.

Nous décrivons maintenant la façon dont les actions des différents agents sont ordonnancées. Une action est dite *autorisée* si et seulement si sa garde est vraie. Un agent est dit *activable* si et seulement si il est en transit dans une arête ou si au moins une de ses actions est *autorisée*. Par extension, un nœud ou une arête est dit *activable* si et seulement si il ou elle contient au moins un agent *activable*. L'activation d'une arête ou d'un nœud consiste en l'activation de l'agent qui y est situé. Quand un agent est activé sur une arête, il se déplace de manière *atomique* sur le nœud destination de l'arête. Notons qu'il n'y a aucune garantie que les agents sortent des arêtes en respectant l'ordre dans lequel ils y sont entrés. L'activation d'un agent sur un nœud consiste en l'exécution *atomique* des instructions d'une de ses actions *autorisées*. Les *activations* sont gérées par un *oracle* que nous considérons ici *distribué* et *faiblement équitable*. « Distribué » signifie que, tant qu'il existe des agent activables, l'oracle active à chaque étape atomique un sous-ensemble non vide d'arêtes et de nœuds activables. « Faiblement équitable » signifie que chaque agent continuement activable est activé en un temps fini.

Nous décrivons maintenant le comportement de notre algorithme. Dans ce qui suit, nous appelons *foyer* d'un agent, le nœud sur lequel cet agent apparaît pour la première fois. Nous appelons *propriétaire* l'agent unique qui a l'étiquette minimale. Tous les autres agents sont appelés *locataires*. Le principe de l'algorithme est de réunir tous les agents sur le nœud foyer du propriétaire.

Notre algorithme est organisé en deux phases : une phase de *parcours* et une phase d'*assemblage*. Tout agent commence par la phase de *parcours* en vue de construire un arbre couvrant enraciné en son foyer et d'écrire son étiquette sur tous les tableaux. Un agent passe en phase d'assemblage dès lors qu'il se rend compte qu'il n'est pas le propriétaire. Il se met alors à suivre les arêtes de l'arbre construit par celui qu'il pense être le propriétaire en vue d'atteindre son foyer. Le propriétaire quant à lui réalise toujours complètement sa phase de parcours. A la fin de son parcours, il s'arrête définitivement à son foyer. La phase d'*assemblage* n'est donc exécutée que par les agents locataires.

Plus précisement, le propriétaire construit un arbre couvrant enraciné en son foyer en écrivant dans les variables *est_foyer*, *courant* et *P* des tableaux de chaque nœud. Initialement, chaque agent ne sait pas s'il est le propriétaire ou non, il ne connaît ni l'étiquette du propriétaire, ni le foyer de ce dernier. Ainsi, chaque agent se comporte comme s'il était le propriétaire jusqu'à ce qu'il découvre qu'il ne l'est pas. Par conséquent, tant qu'il ne découvre pas de nœud marqué avec une étiquette plus petite que la sienne, un agent écrit son étiquette dans la variable *P* du tableau de chaque nœud qu'il visite. Dans le cas contraire, il réalise alors qu'il n'est pas le propriétaire et se met à exécuter la phase d'assemblage. Il utilise alors les informations contenues dans les tableaux pour suivre un chemin de l'arbre couvrant de l'agent qu'il prend pour être le propriétaire jusqu'au foyer de celui-ci.

Remarquons qu'un agent locataire peut commettre des erreurs, soit en considérant à tort un autre locataire avec une étiquette plus petite que la sienne comme étant le propriétaire, soit en suivant l'arbre couvrant du propriétaire en cours de construction. Dans ce dernier cas, il est possible que l'agent suive un cycle. Cependant, une fois que la phase de parcours du propriétaire est terminée, plus aucune erreur n'est possible. En effet, à chaque fois qu'un agent est activé sur un nœud tel que la variable *P* contient une étiquette supérieure à la sienne, il considère le nœud comme non visité et écrase les données dans le tableau avec les siennes. Par conséquent, l'étiquette du propriétaire finit par être disponible sur tous les tableaux (les écritures du propriétaire ne sont jamais effacées par les locataires). De cette manière, tous les locataires finissent par connaître l'étiquette du propriétaire.

Puisque les écritures du propriétaire ne sont jamais effacées, la construction de son arbre couvrant s'opère en un temps fini. Le propriétaire reste alors sur son foyer pour toujours et son arbre couvrant est décrit par les tableaux des nœuds. Ainsi, tous les locataires finissent par atteindre le foyer du propriétaire en utilisant les données distribuées dans les tableaux et le rendez-vous a bien lieu.

Constantes d'un agent a :
$\mathit{étiquette}_a$: étiquette de l'agent.
Primitives pour un nœud v :
`Aller_vers`(arête) — Envoie l'agent sur l'arête spécifiée.
`Vient_de`$() \in \{0,1,\ldots,\delta_v-1\} \cup \{\bot\}$ — Retourne l'indice de l'arête par lequel l'agent est arrivé, $\bot$ sinon.
Variables d'un nœud v :
$\mathit{courant} \in \{0,1,\ldots,\delta_v-1\} \cup \{\bot\}$ — Arête explorée par l'hôte local, initialisée à $\bot$.
$\mathit{est\_foyer}$: booléen — dit si le nœud est actuellement considéré comme foyer par un agent, initialisée à faux.
P : contient une étiquette — La plus petite étiquette d'un agent ayant visité ce nœud, non initialisée.

Macros pour un agent a sur le nœud v :

`Explore`(arête)	=	$\mathit{courant} \leftarrow$ arête ; `Aller_vers`(arête)
`Suivant`()	=	(`Vient_de`$()+1$) mod δ_v

Prédicats pour un nœud v :

$\mathit{Démarrage}$	$\equiv$	`Vient_de`$() = \bot$
$\mathit{Nouveau}$	$\equiv$	$\mathit{courant} = \bot \vee \mathit{étiquette}_a \prec P$
Retour	$\equiv$	$\mathit{courant} \neq \bot \wedge P = \mathit{étiquette}_a \wedge$ `Vient_de`$() \neq \bot$
$\mathit{EtreLocataire}$	$\equiv$	$\mathit{courant} \neq \bot \wedge P \prec \mathit{étiquette}_a$
Cycle	$\equiv$	$\mathit{courant} \neq$ `Vient_de`$()$
$\mathit{RendezVous}$	$\equiv$	$\mathit{est\_foyer} = \mathit{vrai}$
Fin	$\equiv$	$\mathit{RendezVous} \wedge$ (`Suivant`$() = 0$)

Commandes gardées pour un agent a sur un nœud v :

Démarrer	:	$\mathit{Nouveau} \wedge \mathit{Démarrage}$	$\longrightarrow$	$P \leftarrow \mathit{étiquette}_a$; $\mathit{est\_foyer} \leftarrow \mathit{vrai}$; `Explore`(0)
ExplorerNouveau	:	$\mathit{Nouveau} \wedge \neg\mathit{Démarrage}$	$\longrightarrow$	$P \leftarrow \mathit{étiquette}_a$; $\mathit{est\_foyer} \leftarrow \mathit{faux}$; `Explore`(`Suivant`())
ExplorerSuivant	:	$\mathit{Retour} \wedge \neg\mathit{Cycle} \wedge \neg\mathit{Fin}$	$\longrightarrow$	`Explore`(`Suivant`())
DéfaireCycle	:	$\mathit{Retour} \wedge \mathit{Cycle}$	$\longrightarrow$	`Aller_vers`(`Vient_de`())
Assemblage	:	$\mathit{EtreLocataire} \wedge \neg\mathit{RendezVous}$	$\longrightarrow$	`Aller_vers`($\mathit{courant}$)

FIGURE 1: Algorithme *RDV* : rendez-vous entre plusieurs agents dans un graphe non orienté.

Nous détaillons maintenant les comportements respectifs du propriétaire et des locataires.

Comportement du propriétaire. Le propriétaire h sait qu'il est activé pour la première fois grâce à la fonction `Vient_de`() qui retourne $\bot$. Il marque alors son foyer en écrivant sur le tableau : les variables *est_foyer* et P prennent respectivement les valeurs *vrai* et $\mathit{étiquette}_h$. Puis, partant de son nœud foyer, h réalise un parcours en profondeur du graphe afin de construire l'arbre couvrant et diffuser son étiquette. Pour ce faire il initialise la variable *courant* du tableau à 0 et quitte le nœud par l'arête d'indice 0. Quand il arrive sur un nœud qu'il n'a jamais visité, le tableau est vide ($P = \bot$) ou il a déjà été écrit par un locataire g (c'est-à-dire $P = \mathit{étiquette}_g$ avec $\mathit{étiquette}_h \prec \mathit{étiquette}_g$). Dans les deux cas, h modifie le tableau en affectant son étiquette $\mathit{étiquette}_h$ à la variable P et la valeur faux à *est_foyer* (cette dernière affectation permet d'effacer les foyers des locataires). h poursuit alors son parcours en affectant *courant* à `Vient_de`$()+1$ (mod δ_v) et quitte le nœud par l'arête pointée par la variable *courant*. Lorsqu'il arrive sur un nœud qu'il a déjà visité (c'est-à-dire un nœud pour lequel $P = \mathit{étiquette}_h$), soit $\mathit{courant} \neq$ `Vient_de`(), soit $\mathit{courant} =$ `Vient_de`(). Dans le premier cas, h a suivi un cycle et il retourne alors au nœud précédent (en utilisant `Vient_de`()). Dans le second cas, h est revenu en arrière car le parcours depuis l'arête dont il vient est terminé. Il incrémente alors la variable *courant* et quitte le nœud par l'arête d'indice *courant* de façon à poursuivre son parcours. En suivant cette méthode, h va nécessairement terminer son parcours sur son foyer : quand il arrive sur un nœud v où $P = \mathit{étiquette}_h$, $\mathit{est\_foyer} = \mathit{vrai}$, et $\mathit{courant} = \delta_v - 1$. L'arbre couvrant enraciné en son foyer est alors décrit par les variables *courant* des tableaux et chaque variable P vaut $\mathit{étiquette}_h$.

Comportement d'un locataire. Un agent locataire g quelconque agit comme l'agent propriétaire jusqu'à ce qu'il soit activé sur un nœud où $P \prec \mathit{étiquette}_g$. L'agent g passe alors à sa phase d'*assemblage* : si *est_foyer* est faux, il quitte le nœud via l'arête indiquée par la variable *courant*. Dans le cas contraire, il reste sur le nœud en question qui peut être le foyer du propriétaire. Ainsi, une fois le parcours du propriétaire terminé, les locataires remontent dans l'arbre couvrant du propriétaire à chaque activation jusqu'à atteindre son foyer et s'arrêter définitivement.

Il faut noter qu'un locataire peut arriver sur un nœud marqué par un locataire ayant la même étiquette. Etant amnésique, le locataire n'a aucune raison de croire qu'il n'est pas sur un nœud qu'il a déjà visité. Cette situation ne pose pas de problème car chaque nœud finit par être visité par le propriétaire dont l'étiquette est unique.

Une *ronde* correspond au temps nécessaire pour que l'agent activable le plus lent agisse. Ainsi, la première ronde d'une exécution termine dès lors que tous les agents activables au temps 0 ont agi. Chaque ronde suivante est construite de la même manière en considérant le suffixe de l'exécution suivant la ronde précédente. Dans [CDPR09], nous avons démontré le résultat suivant :

Théorème 1 *L'algorithme RDV réalise un rendez-vous en $\Theta(m)$ rondes où m est le nombre d'arêtes.*

3 Optimalité et nécessité des hypothèses

La première borne d'optimalité concerne la complexité en temps. Dans [CDPR09], nous montrons que tout algorithme de rendez-vous déterministe nécessite qu'un agent au moins explore tout le graphe. Un tel parcours est réalisé en $\Omega(m)$ rondes où m correspond au nombre d'arêtes du graphe. Ainsi, d'après le théorème 1, nous pouvons conclure que notre algorithme est asymptotiquement optimal en nombre de rondes.

En ce qui concerne l'optimalité en espace, nous montrons dans [CDPR09] que tout algorithme déterministe de rendez-vous réalisé dans notre modèle nécessite $\log(\delta_v+1)+\log(\mathcal{L}_{\max})+1$ bits de mémoire dans le tableau de chaque nœud v où δ_v est le degré de v et $\mathcal{L}_{\max}$ est le nombre maximal d'étiquettes disponibles. L'idée de la preuve de cette borne est résumée par les trois points suivants :

1. Tout d'abord, nous savons qu'un agent au moins doit réaliser un parcours déterministe du graphe. Cet agent étant amnésique, il doit pouvoir stocker dans le tableau de chaque nœud v les informations lui permettant de savoir s'il a visité le nœud et quelles arêtes de ce nœud il a déjà traversé. Pour cela, $\log(\delta_v+1)$ bits sont nécessaires sur le tableau de v.
2. Ensuite, on a besoin d'au moins un bit pour distinguer le point de rendez-vous parmi les nœuds (anonymes).
3. Enfin, il est nécessaire de stocker dans le tableau de chaque nœud l'étiquette de l'agent qui décide du point de rendez-vous, c'est-à-dire, $\log(\mathcal{L}_{\max})$ bits.

Notre algorithme utilisant exactement $\log(\delta_v+1)+\log(\mathcal{L}_{\max})+1$ bits de mémoire sur le tableau de chaque nœud, nous obtenons donc :

Théorème 2 *L'algorithme RDV est optimal en espace et asymptotiquement optimal en rondes.*

Nous justifions maintenant que les hypothèses que nous avons ajoutées au modèle de départ sont nécessaires.

Tout d'abord, nous savons de [DMGK$^+$06] que l'utilisation d'étiquettes et de tableaux est nécessaire pour résoudre le problème de manière déterministe avec des agents amnésiques. Ensuite, l'existence d'une étiquette minimale affectée à un seul agent est également nécessaire. En effet, en la supprimant, il serait impossible de particulariser au moins un agent. Or, l'obtension d'une solution déterministe est subordonnée à l'existence d'un unique agent décidant d'un point de rendez-vous. Enfin, en supprimant les indices locaux des arêtes, le parcours des agents ne pourrait pas être déterministe.

Il est donc impossible de supprimer une de ces hypothèses pour que le problème du rendez-vous puisse être résolu de manière déterministe dans notre modèle.

4 Conclusion et Perspectives

Nous avons proposé un algorithme déterministe de rendez-vous pour des agents amnésiques évoluant dans un graphe non orienté anonyme quelconque. Cet algorithme est optimal en espace et asymptotiquement optimal en nombre de rondes.

Nous allons maintenant porter nos efforts sur le rendez-vous dans les graphes orientés connexe. Dans de tels graphes, un nœud ne pouvant pas toujours être atteignable à partir d'un autre, le rendez-vous n'est pas toujours réalisable. Nous tenterons alors de répondre à la question suivante : ≪ *Quelle est la classe maximale des graphes qui admettent une solution* ? ≫

Références

[CDPR09] Fabienne Carrier, Stéphane Devismes, Franck Petit, and Yvan Rivierre. Space-optimal deterministic rendezvous. In *PDCAT*, pages 342–347, 2009.

[DMGK$^+$06] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theor. Comput. Sci.*, 355(3) :315–326, 2006.

[YY96] Xiangdong Yu and Moti Yung. Agent rendezvous : A dynamic symmetry-breaking problem. In *ICALP '96 : Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, pages 610–621, London, UK, 1996. Springer-Verlag.